SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. **10**, 395–414 (1998)

Research

Knowledge Base Clustering for KBS Maintenance

OOK LEE[1]* and PAUL GRAY[2]

[1]*Department of Business Administration, Hansung University, Seoul, Korea*
[2]*Department of Information Science, Claremont Graduate University, Claremont CA 91711–6160, U.S.A.*

SUMMARY

Clustering the rules in a knowledge-based system (KBS) on the basis of their static distance (lexical information) reduces the burden of understanding in the mind of the maintainer. This paper explores a clustering algorithm based on the Hopfield neural net algorithm that clusters automatically using lexical similarity. Using that algorithm, this paper presents a tool that can aid the maintainer in maintaining a KBS. The tool is the Rule Base Clusterizer (RBC) which structures the KBS rule base to make it appear easy to understand for the maintainer. This paper shows by using entropy and Miller's number that the RBC finds the best clustering that produces both an adequate amount of information and acceptable sized clusters. The paper also presents three examples of running the RBC on real-world rule bases.

KEY WORDS: knowledge base system maintenance; software engineering; neural network; rule base maintenance; rule clustering; maintenance tools

1. INTRODUCTION

The content of a knowledge base offers few clues to the maintainer of knowledge-based systems (KBS) compared with conventional software maintenance. In conventional software maintenance, the maintainer can utilize salient features and landmarks of the program code as long as the maintainer is familiar with the programming language. For example, the control statements such as FOR, DO WHILE, BEGIN, END or PROCEDURE are good landmarks to distinguish a code section while reading the code. Such features help in building the maintainer's mental model of the program code. In programming languages supporting structured programming, the maintainer is able to predict the organization of the content. For example, the maintainer can recognize that a module is called by another module by simply reading the code. A structured programming language lets the developer write code organized in modules, which in turn, makes the maintenance work easier by facilitating program comprehension.

However, the rules of a knowledge base have no salient features or landmarks—i.e.,

* Correspondence to: Dr. Ook Lee, Department of Business Administration, Hansung University, 389, 2 ga. Samsun-dong, Sungbuk-gu, Seoul, Korea. Email: leeo@hansung.ac.kr

CCC 1040–550X/98/060395–20$17.50

Received 11 March 1997
Revised 23 June 1998

every line of code is composed of the same structure which is IF ... THEN .... Particularly in production systems that use a rule structure for knowledge representation, the monotonous surface of the content of the knowledge base is a serious obstacle to the maintainer who has to understand the content of the knowledge base before undertaking any maintenance action.

KBS maintenance is difficult partly due to the relative lack of software tools, such as on aiding knowledge base understanding, that can facilitate the maintenance process. The goal of this paper is to develop an algorithm to facilitate the understanding of the rules of a knowledge base, an algorithm that can be used as the basis for a tool in doing KBS maintenance.

The algorithm explored in this paper structures the unstructured rule base by clustering rules using a neural network algorithm based on a Hopfield net. Other alternatives to the Hopfield net could be conventional clustering algorithms without using inductive learning technique and those used in text classification (Chen *et al.*, 1994, pp. 21–22, 115–119). Those alternative algorithms cannot produce results without supervision. The application of a Hopfield net as described here can work without supervision by selecting an optimal criterion value using a juxtapositioning of entropy and Miller's number. Among neural net techniques, the Hopfield net is better suited since it does not require training; clustering is done automatically with the only necessary input being the criterion value that can be derived systematically through juxtapositioning. Thus, an algorithm based on a Hopfield net could be the ideal choice for grouping rules automatically without relying on training cases and supervision.

If a software tool implementing such an algorithm were available, the job of maintaining a KBS could be less onerous since the maintainer can then understand the rule base more easily. The clustered rules provide a starting point for human cognition in understanding the material shown to the maintainer. When lexically similar looking rules are bundled together in proximity, people can find it easier to draw a mental model for the structure of the rule base. For example, in a medical KBS, there could be a number of rules about a specific disease and these rules tend to have lots of common terms in the body of rules. The maintainer will recognize the relatedness of those rules more quickly compared with a situation where these rules are scattered around the entire rule base of more than 10 000 rules. Of course, if the rule base is well written—i.e., written in an organized way as in RIME—rules are already bundled together for the benefit of maintenance (Soloway, Bachant and Jensen, 1987). But, as for rule bases which are not written in a language like RIME that forces bundling of relevant rules together, there has to be a way to organize or structure them, such as clustering them into groups.

Since semantic grouping is always bounded in the particular domain, creating a semantic clusterizer is not very practical. In order to be applied to rule bases from any domain, we have to resort to structuring based on the syntactic information in the rules and the lexical similarity among the rules. Our assumption is that there is a tendency that if we group rules together having many common terms in their body of rules, the groups individually and collectively would represent a semantically meaningful structure.

There are practical reasons supporting the assumption that clustering based on lexical similarity of rules would make understanding of the rule base easier. Once rules in a rule base are grouped together, we know that a particular group contains rules with many

common terms. Thus, when the maintainer wants to do maintenance, such as the modification of a term in a rule, the maintainer can save time and effort by knowing that rules with the term to be modified are concentrated in the same group. The maintainer can fix most of them there on the spot instead of having to search the entire rule base repeatedly to find and modify each instance.

Usually KBS maintenance actions include modification, deletion and addition of terms in a rule (Coenen and Bench-Capon, 1993). These actions require a secondary wave of fixing all 'relevant' rules. Clustering based on lexical similarity offers an economical way of finding 'relevant' rules because the rules that have more terms in common should be fixed together. In this way, not only can the maintainer increase his or her understanding of the rule base structure, but also the maintainer can use the clusters of rules in actual maintenance actions for saving time and effort. In summary, clustering based on lexical similarity can aid the understanding of a rule base and provide assistance in performing actual maintenance action.

2. RULE BASE CLUSTERING

2.1. Program comprehension versus rule comprehension

Program comprehension has long been identified as a major factor in the effective maintenance of conventional software systems. Sharon (1996) pointed out that excessive time is spent learning and figuring out the program source code. In other words, the maintainer of conventional software spends significant time just to figure out what the code is all about. This analysis step makes the maintenance process less efficient. To increase efficiency, software tools have been introduced to aid in program comprehension. Thus, such tools as a program scanner, program navigator and logic flow tracer, trace the logic and data flows and allow browsing within and among system components. Since rule comprehension is as important in KBS maintenance as program comprehension is in conventional software maintenance, a tool that facilitates the understanding of rules should be beneficial.

What makes rule comprehension easier should be identified first before embarking on constructing a tool to aid in KBS rule comprehension. We can utilize gains from research on program comprehension in conventional software maintenance. Von Mayrhauser and Vans (1995) identify the understanding mechanism they term chunking as an important element of the mental model. Chunking creates new, higher-level abstract structures from chunks of lower-level structures. That is, lower-level structures can be combined into larger structures at a higher abstraction level. Chunking is what a human brain does to the program mentally, which is equivalent to a clustering of the program components. If a program is already clustered by, for example, modularization, the maintainer does not have to spend as much time and effort in chunking when creating a mental model. An initial mental model of the program is established with ease by simply accepting the clustering of the program. If time spent on chunking is reduced, the maintenance task can be facilitated. In KBS maintenance, the maintainer must do chunking to understand the knowledge base. Rules are likely to be more difficult to chunk than program code

since there are no lexical structures that can play the role of landmarks. Thus, if the rules are clustered, the maintainer can accept the clusters as an initial chunking when building a mental model.

The principle applied in this paper is that clustering that minimizes the cognitive load in mental model building facilitates rule understanding as well as program understanding. Both can lead to more effective maintenance. A software tool that clustered rules in a rule base would therefore be useful in KBS maintenance by facilitating the building of mental models. Such a tool should be able to accept input regardless of domain so that the maintainer of existing systems can use it for facilitating the understanding of nearly any rule base.

2.2. Constructing the rule base

The concept of clustering rules can be understood from the following examples. Assume that a KBS is written in VP-Expert (Pigford and Bauer, 1995, pp. 187–193). In VP-Expert, a **term** as a name appearing in the rule excluding identifiers such as **IF**, **AND**, **OR**, **THEN**, =, +, –, or *. The following is a sample of a rule base expressed in VP-Expert:

Rule 1:
IF Functionality = Simple
AND Cost = Low
THEN Word-Processor = Product-A;
Rule 2:
IF Functionality = Complex
AND Cost = Low
THEN Word-Processor = Product-B;
Rule 3:
IF Functionality = Simple
AND Cost = High
THEN Word-Processor = Product-C;

Notice that these three rules are rules for selecting a word processor—i.e., this is a cluster for selecting a word processor. In the above rules, the terms are ‘Functionality, Simple, Cost, Low, High, Word-Processor, Product-A, Product-B, Product-C’. In addition to these rules, the following rule could exist together:

Rule 4:
IF Personality = good
AND Experience = Much
THEN Candidate = Jefferson;
Rule 5:
IF Personality = bad
AND Experience = Less
THEN Candidate = Burr;
Rule 6:

IF Personality = good
AND Experience = Less
THEN Candidate = Adams;

Notice that these rules form a cluster for selecting a presidential candidate. In the above rules, the terms are 'Personality, Good, Experience, Much, Bad, Less, Candidate, Adams, Burr, Jefferson'. When these two distinctive clusters exist in a rule base without explicit landmarks, the maintainer could be confused and would spend some time to discover the existence of two distinctive modules in a rule base. Furthermore, there is also a chance that the rules could be mixed up. For example, this rule base could be organized in the following order without violating the syntax of the programming language:

[Rule 1, Rule 4, Rule 3, Rule 5, Rule 2, and Rule 6]

When a mixed-up organization has resulted from undisciplined previous development and maintenance work, the maintainer will spend extra time in understanding the rules. To facilitate maintenance in such situations, an automatic rule clusterizer would be helpful that can turn a mixed-up rule base or a somewhat structured one, into a distinctively structured rule base where all the rules are categorized into clusters. For example, where [ ] identify clusters, a rule base tool should categorize the above rule base into two clusters:

[Rule 1, Rule 2, Rule 3]

[Rule 4, Rule 5, Rule 6].

With clustering such as this, the maintainer would be able to reduce time spent in comprehending the rule base. This approach is particularly helpful in the maintenance of very large rule bases.

2.3. Static distance

Rules can be characterized in terms of their 'static distance' from one another. Here 'static' refers to the idea that the distance is fixed in terms of the syntactic structure of the rule base. We define the static distance d between two rules, i,j, to be used in automatic clusterization as follows:

$$d_{ij} = 1 - \frac{CountofIdenticalTermsInRules(i)and(j)}{\min(CountofTermsInRule(i),\ CountofTermsInRule(j))} \tag{1}$$

For example, as shown in Figure 1, the 0·33 static distance between Rule 1 and Rule 2 comes from expression (1): $d_{12} = 1 - 4/6 = 1/3$. The four vertical lines in Figure 1 point out the identical terms in the two rules, Rule 1 and Rule 2.

The static distances of the rules in our six-rule sample rule base can be shown as a matrix as in Table 1. A static distance matrix contains quantities between 0 and 1, with

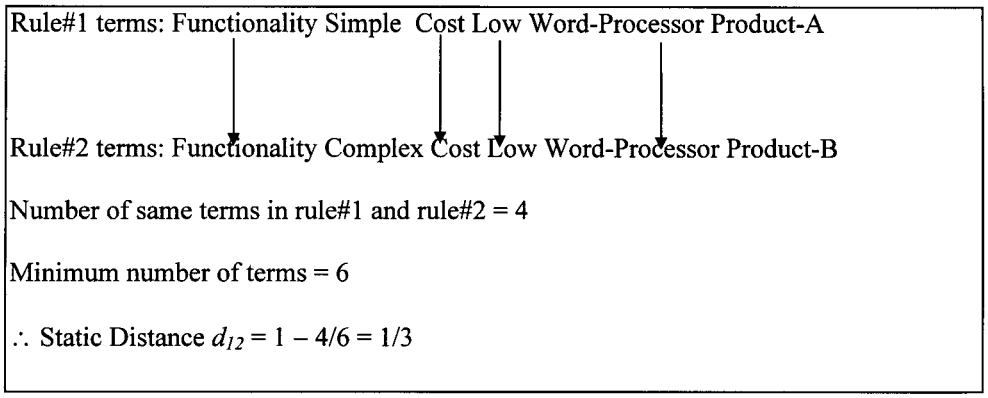

Figure 1. An example of static distance calculation

Table 1. Static distance matrix for six-rule example rule base

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Rule 1	0	0·33	0·33	1	1	1
Rule 2	0·33	0	0·5	1	1	1
Rule 3	0·33	0·5	0	1	1	1
Rule 4	1	1	1	0	0·5	0·33
Rule 5	1	1	1	0·5	0	0·33
Rule 6	1	1	1	0·33	0·33	0

0 on the diagonal. A quantity of 1 indicates no relation. In Table 1, the 3×3 blocks of 1's show the independence of the two clusters.

This matrix representation, which is symmetric, can also be described as a network of nodes and weighted links in a neural net. This suggests the use of the Hopfield net to classify rules automatically for clustering the rule base. Based on the static distance matrix such as the one shown in Table 1, a Hopfield matrix can be created and implemented. The Hopfield net converges after a series of iterations. The converged Hopfield net produces the clusters for 'related' rules.

2.4. Hopfield neural net

Among the many variations of neural nets, one of the simpler algorithms is the Hopfield net, which is useful in pattern recognition where neurons are fully connected to other neurons. The Hopfield net algorithm is well described in many textbooks on neural networks; our implementation was based on the algorithm presented in Fausett (1994, pp. 79–83). We implemented the algorithm in C++ on IBM PC-compatible machines, in Pascal on VAX VMS minicomputers, and in GNU C on Sun/UNIX Workstations. The reason for multiple implementations was due to the large amount of memory and CPU

time that this algorithm took. We experimented with different hardware and different implementations seeking faster speed.

We used a fully interconnected Hopfield neural net in the sense that each unit is connected to every other unit. The net has symmetric weights with no self-connection—i.e., where W represents a weight, then $W_{ij} = W_{ji}$ and $W_{ii} = 0$. Since the discrete Hopfield net requires binary inputs, we transformed the static distance matrix into a binary one which we called the Hopfield matrix.

We created a Hopfield matrix by setting up a criterion value that determines which binary value should be used. For example, when we set the criterion value as 0·7, every non-diagonal static distance value which is less than 0·7 is replaced with a binary value 1 while the static distance value which is equal to or larger than 0·7 is replaced with a binary value −1. In other words, rules whose static distance to a given rule is smaller than the chosen criterion value are considered to be 'close in distance'. Close-in-distance rules can be candidates to be in the same cluster with the given rule. Rules whose static distance to a given rule is equal to or larger than the criterion value are not candidates to be in the same cluster. In order to make a Hopfield matrix, there has to be a threshold value to determine which rules are 'in' and which ones are 'out'. This is why a positive binary value 1 is assigned to distances less than a criterion value and a negative binary value −1 to distances larger or equal to a criterion value. The diagonal value of the Hopfield matrix remains 0 as in the distance matrix.

As the criterion value changes in the range from 0 to 1, the Hopfield matrix changes. For instance, a change takes place when the criterion value increases so that it includes one or more additional distances d_{ij}. In the example in Table 1, if the criterion value is 0·33, then all distances less than 0·33 in the distance matrix will be replaced with 1 while the rest will be −1. Thus, all criterion values less than 0·33 will produce the same Hopfield matrix out of the distance matrix. Only if the criterion value is greater than 0·33 is a different Hopfield matrix obtained. These distances act as the 'change points' in the criterion values. Rather than using criterion values from 0 to 1 by incrementing by some arbitrary amount, say, 0·01, we can use these change points as the criterion values used to produce clusterings. In practice, the distances, d_{ij}, of the distance matrix are sorted in ascending order from 0 to the biggest distance $\leqslant 1$ without any duplication. These distances become the change points for clustering. In Table 1, we observe the following change points: 0, 0·33, 0·5 and 1. Table 2 is the Hopfield matrix representation of Table 1 for all criterion values between 0·5 and 1.

Table 2. Hopfield matrix for six-rule example with criterion values $\geqslant 0.5$

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Rule 1	0	1	1	−1	−1	−1
Rule 2	1	0	1	−1	−1	−1
Rule 3	1	1	0	−1	−1	−1
Rule 4	−1	−1	−1	0	1	1
Rule 5	−1	−1	−1	1	0	1
Rule 6	−1	−1	−1	1	1	0

Notice that this matrix fits the required properties of the Hopfield net. That is, it is symmetric and has 0's on the diagonal. Thus, this net, which was created out of the static distances between rules, follows the proven laws of the Hopfield net such as convergence to a stable limit point after a finite number of iterations (Fausett, 1994, pp. 79–83). The implementation algorithm for the discrete Hopfield net, which is based on Fausett (1994, pp. 79–83), is summarized below.

Hopfield net algorithm

Step 0. Create the Hopfield matrix, **W**, based on a static distance matrix, a square matrix of n rules. Before creating the Hopfield matrix, choose a criterion value that can be any number from 0 to 1.

Step 1. For each input vector **x**, do Steps 2 to 6 where the input vector **x** is the indicator for the given rule. This input vector of length n has 1 only in the given rule number's place and 0's in every other place. For example, if the given rule is Rule 2 in the six-rule example, vector **x** will be [0 1 0 0 0 0].

Step 2. Set the initial value of the activations vector, **y**, to be a copy of the input vector, **x**. (Activations in a neural net are output signals that are produced by the net and fed back to the net.)

Step 3. Do Steps 4 to 6 for each element of vector $\mathbf{y}_i$.

Step 4. Where $\mathbf{w}_{ij}$ are the elements of the ith column of the Hopfield matrix from step 0, compute the net input for the ith element in the **y** vector for Rule j by summing products formed from column j:

$$\mathbf{y}_i = \mathbf{x}_i + \Sigma \mathbf{y}_j \mathbf{w}_{ji} \tag{2}$$

Step 5. Determine activation (output signal) for the ith element in the **y** vector for Rule j as follows:

$$\mathbf{y}_I = 1 \text{ if } \mathbf{y}_i > 0 \text{ or } \mathbf{y}_I = 0 \text{ if } \mathbf{y}_I < 0 \tag{3}$$

Note that when $\mathbf{y}_i = 0$, $\mathbf{y}_i$ does not change.

Step 6. Update the activation vector **y** for Rule j with the new value of $\mathbf{y}_i$.

Step 7. Output the **y** vector as the cluster to which the given **x** vector (rule number) belongs.

The result of applying the Hopfield net algorithm on the previously presented six-rule example rule base is as follows, where the distance matrix, **d**, is as shown in Table 1, and the Hopfield matrix, **W**, is as shown in Table 2:

Rule 1

Step 1: The **x** vector for Rule 1 is [1 0 0 0 0 0].

Step 2: The **y** vector for Rule 1 is [1 0 0 0 0 0].

Step 3: The first iteration processes Rule 1 data.

Step 4: From expression (2) the product of the $\mathbf{y}_1$ vector [1 0 0 0 0 0] and the $\mathbf{w}_1$

column vector [0 1 1 −1 −1 −1] is [0 0 0 0 0 0]. The sum of those products is 0. Since the starting value of $\mathbf{y}_1$ was 1, adding 0 to it yields a value of 1.

Step 5: From expression (3), since the former value was 1 and the computed value was 1, the new value is 1.

Step 6: Substituting the new value from Step 4 into the **y** vector resulting from the previous iteration of [1 0 0 0 0 0] yields a **y** vector of [1 0 0 0 0 0], which in this instance is no net change.

Step 3: The second iteration processes Rule 2 data.

Step 4: From expression (2) the product of the $\mathbf{y}_1$ vector [1 0 0 0 0 0] and the $\mathbf{w}_1$ column vector [1 0 1 −1 −1 −1] is [0 0 0 0 0 0]. The sum of those products is 1. Since the starting value of $\mathbf{y}_1$ was 0, adding 1 to it yields a value of 1.

Step 5: From expression (3), since the former value was 0 and the computed value was 1, the new value is 1.

Step 6: Substituting the new value from Step 4 into the **y** vector resulting from the previous iteration of [1 0 0 0 0 0] yields a **y** vector of [1 1 0 0 0 0].

Step 3: The third iteration processes Rule 3 data. Doing Steps 4 to 6 yields a **y** vector of [1 1 1 0 0 0].

Step 3: The fourth iteration processes Rule 4 data. Doing Steps 4 to 6 yields a **y** vector of [1 1 1 0 0 0].

Step 3: The fifth iteration processes Rule 5 data. Doing Steps 4 to 6 yields a **y** vector of [1 1 1 0 0 0].

Step 3: The sixth iteration processes Rule 6 data. Doing Steps 4 to 6 yields a **y** vector of [1 1 1 0 0 0].

Continuing the algorithm from Step 1 for Rule 2 and Rule 3 also results in **y** vectors of [1 1 1 0 0 0].

Continuing the algorithm from Step 1 for Rule 4, Rule 5 and Rule 6 results in **y** vectors of [0 0 0 1 1 1].

Table 3 summarizes the results of applying the Hopfield net algorithm to the six-rule example. Note that the resulting square matrix is symmetric along the diagonal.

2.4. Mutually exclusive clustering of a rule base

The final converged status of a rule vector represents a pattern that is unique to the particular rule—i.e., a rule cannot have more than one vector pattern as a result of

Table 3. Clusters indicated by application of the Hopfield net algorithm

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Rule 1	1	1	1	0	0	0
Rule 2	1	1	1	0	0	0
Rule 3	1	1	1	0	0	0
Rule 4	0	0	0	1	1	1
Rule 5	0	0	0	1	1	1
Rule 6	0	0	0	1	1	1

applying the Hopfield algorithm. This is an obvious property of the algorithm that produces one final resulting vector pattern after doing all of the iterations. Thus, when each rule is given a unique vector pattern as shown previously, mutually exclusive clustering of rules is possible based on the vector patterns of each rule. In other words, we can group rules that have the same vector pattern into one cluster. In this way, we can cluster all rules in the rule base into mutually exclusive clusters. These clusters are mutually exclusive since no single member (rule) of a cluster can be a member (rule) of another cluster. Thus, Table 3 indicates the presence of two clusters of rules.

3. ANALYSIS OF RULE BASE CLUSTERING

3.1. Evidence of usefulness

Depending on the criterion value chosen (it must be in the range from 0 to 1), rule base clustering can produce very different results. Thus we have to find the criterion value which produces the most acceptable clustering of a rule base. The clustered rule base should be analysed in order to demonstrate its usefulness in aiding KBS maintenance. While the usefulness of a clustered rule base is clear as explained previously, there has to be a theoretical ground to demonstrate that rule base clustering on a particular criterion value is most useful in KBS maintenance. For this purpose, we use the concept of the entropy of a rule base. This concept provides a measure of the quantity of information in the clustered rule base. Also, we use the Miller's magic number concept to assess how much information overload is contained in the clustered rule base for the maintainer's cognition.

3.2. Entropy of a rule base

In thermodynamics from which the concept originates, the entropy function is a measure of the degree of disorder in certain states of nature. In information economics, it can be used to measure the quantity of information. From Ahituv and Neumann (1990, pp. 55–60), the general formula for the entropy, H, of n mutually exclusive events whose probabilities of occurrence individually are $P_1, \ldots, P_n$ and collectively are equal to 1, is:

$$H = -\sum_{i=1}^{n} (\mathrm{P}_i \log_2 P_i) \tag{4}$$

Thus, $H = 0$ (the minimum value of the entropy function) means that there is no information in the rule base—i.e., a rule base in which all rules fall into a single cluster has no information which can be beneficial to KBS maintenance. As the entropy increases, the quantity of information in the rule base increases. For example, a clustered rule base contains useful information for KBS maintenance such as relationships among rules. As the number of clusters increases, the entropy value increases and hence the quantity of information about relationships among rules increases.

In rule base clustering, the clusters are mutually exclusive. Where i is the cluster number, we defined the probability, P_i, of a cluster in a rule base as the ratio of the number of rules in the ith cluster to the total number of rules in the rule base. This definition gives an easily calculated probability and satisfies the condition that the summation of the probabilities P_i equals 1.

Since using different criterion values yields different clusterings, the entropy values of different clusterings represent different quantities of information in the rule base. Even though high entropy values mean high quantities of information contained in the particular clustering of the rule base, this is not necessarily good news to the KBS maintainer. This is because high quantities of information about relationships among rules could be confusing and redundant in understanding the rule base. In other words, high entropy can mean information overload for the KBS maintainer. For example, the maximum entropy value of a rule base is $\log_2 N$, which occurs when the criterion value is 0. In this case, all N rules in the rule base are independent and clustering does not give useful information to the KBS maintainer. A clustered rule base with maximum entropy is, in fact, an unclustered one from the maintainer's viewpoint, which yields a minimum of useful information to the maintainer. As the criterion value increases, the maintainer starts to get useful information. But as the criterion value approaches 1, the number of rules in a cluster becomes too big and the maintainer finds the usefulness of the clusters to be decreasing for KBS maintenance. Thus we need to find a criterion value between 0 and 1 that produces an adequate entropy value and that generates a clustering which is most useful to the KBS maintainer.

3.3. Miller's magic number

Miller (1956) claimed that there is a limit on human cognition of objects, which is 7 ± 2, which he obtained through empirical tests. This value is called 'Miller's magic number' or simply 'Miller's number'. That is, human beings have a meaningful chunking size up to 7 ± 2 and chunks whose size is greater than Miller's number are a burden on human cognition. From the size of 1 to 4, there is no burden on cognition; from 5 to 9, there is an increasing probability of being burdensome; from 10 on, there is a burden on cognition. Miller's number can be ambiguous since the size limit can be any value from 5 to 9—i.e., the size limit varies within the small range of 5 to 9. However if a chunk's size is greater than this range, the chunk is cognitively burdensome.

In rule base clustering, we notice that as the criterion value becomes bigger, the number of clusters that have many members in them increases significantly. Any clustering can have a cluster that has more members than Miller's number. Thus, we should try to have a rule base clustering that produces fewer clusters that violate the Miller's number limit in order to create a clustering that is less burdensome to the maintainer's cognition.

As an example, consider a rule base with 32 rules. If the criterion value is set to 0, then all rules are independent and we have 32 clusters of length one rule each. If the criterion value be set to 1, then all rules fall into a single cluster of size 32, which is much larger than Miller's magic number. An ideal clustering would have five clusters consisting of six or seven rules each. In this case Miller's number is achieved and the number of clusters is reasonable.

We defined Miller% as 100 times the ratio of the count number of the clusters with a size greater than seven, to the total number of clusters for this rule base. To be precise, we should look for the number of clusters whose size is greater than 5, 6, 7, 8 and 9, which will produce different Miller% numbers. The number of clusters whose size is greater than five will count in the number of clusters whose size is greater than six, and the same situation exists between Miller's number six and seven, seven and eight, and eight and nine. This indicates that if we chose Miller's number as five, it might be so inclusive that Miller% can be made too big, whereas if we choose Miller's number as nine, it might be so exclusive that Miller% can be made too small. As a compromise, we chose the middle number seven as Miller's number to be used for the Miller% calculation.

Miller% can be described as a measure of information overload in rule base clustering. Miller% ranges from 0 (criterion=0) to 100(criterion=1). Although the lower Miller% is, the better for human cognition, a lower Miller% is not necessarily good from the KBS maintainer's viewpoint. For example, with a very small Miller% each rule could be its own cluster. This certainly should not be a cognitive burden for the maintainer, but it is not informative in terms of aiding KBS maintenance either. A very big Miller% implies that almost all rules are in one or a few clusters, which clearly are too large for human cognition. We therefore have to look for the criterion value that produces a clustering with a Miller% which should not be too big but not too small either in order to make the clusterized rule base relevant to the job of KBS maintenance.

Having some big clusters is acceptable as long as most of the individual clusters do not become too big. Thus, we propose that a cluster whose size is over Miller's number is not always bad. Rather, as clustering is done, grouping of rules should happen rigorously even though some groups get more members than Miller's number. In contrast to Miller's idea, we propose that having a few clusters with a very large amount of information is tolerable because it is also important in maintenance to perceive the big picture of a rule base. Clusters with many members in them can give a clearer picture of the rule base structure to the maintainer.

In summary when the Miller% is small, information overload is small in local understanding, but cluster size is small, which means less help is provided for global understanding. When the Miller% is big, cluster size is large which helps global understanding but creates information overload for local understanding.

3.4. Juxtapositioning of entropy and Miller functions

The problem we faced was to find some middle point in the range of the entropy values as well as the range of the Miller% values to find the most adequate clustering for the rule base in terms of its usefulness to KBS maintenance. This goal could be achieved by a juxtapositioning of the entropy and Miller% functions by using the criterion value as an independent variable. By juxtapositioning, we find that the intersection point of the two functions occurs at a certain criterion value, and this criterion value is hence the one that produces the most adequate clustering of a rule base from the view of both entropy and Miller%.

There are difficulties in choosing the intersection point of entropy and Miller% functions.

- First, there is the problem of multiple intersection points. Since the entropy function

generally decreases with cluster size whereas the Miller% function generally increases and both go to 0 at one end, we can be sure that there exists at least one intersection point of two functions. However, because the two functions do not increase monotonically, multiple intersection points are possible. In the case of multiple intersections, we need to have a conflict resolution strategy.

- Second, since the Miller% and entropy functions are discontinuous, these two functions may not intersect even if it looks like the intersection point exists from drawing a continuous juxtapositioning graph. If we happen to select a criterion value where the Miller% function and entropy functions are equal—i.e., intersection occurs—then there is no problem. But in cases other than that, which is most of the time, we also need to have a conflict resolution strategy. The strategy described below is recommended for choosing the right intersection point.

3.5. Choosing the right intersection point

In a few cases, the intersection will occur at a criterion value. In such a case, we choose the intersection. In most cases, however, the intersection will not occur at one of these criterion values but between adjacent criterion values we denote *a* and *b*. The situation is approximated as follows (see Figure 2).

Consider the interval [*a*, *b*] in the range of [0, 1]. Assume that both the entropy values and the Miller% values have been computed at *a* and at *b*. Call *a* and *b* criterion values.

The subscripts 1 and 2 denote, respectively, the values of the entropy, *e*, and the Miller%, *m*, at criterion values *a* and *b*. For simplicity of explanation, we show a smooth, continuous approximation. In actual cases, the curves may have one or more jumps in an interval.

Although it is possible to approximate the intersection point either by linear interpolation or by interval halving, we argue that (for a given intersection) it is adequate to choose the lower end of the interval [*a*, *b*]. Specifically, e_1, has more information (higher entropy) then e_2 and m_1 has a lower Miller% than m_2. Hence, even though the intersection occurs within the interval, selecting the criterion value at *a* is a reasonable choice.

Two additional situations have to be considered:

(1) The Miller% is 0 at a criterion value *a*, because all the clusters are relatively small. In this case, choose the criterion value *b*.
(2) There are multiple intersection points, such as shown later in Figure 4. Here we

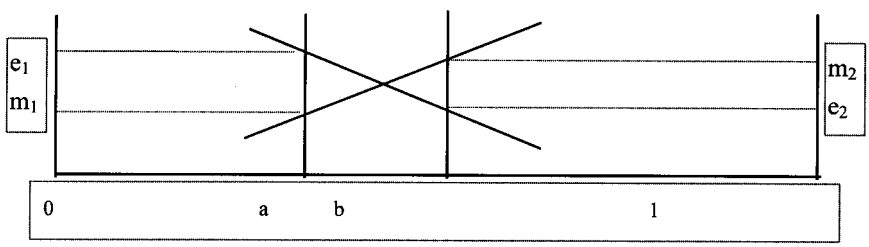

Figure 2. Intersection example

choose the criterion value *a* since, in this case, the information quantity (entropy) is highest and the Miller% is lowest.

4. REAL-WORLD RULE BASE CLUSTERING

4.1. Three examples

This section presents the clustering results from three real-world rule bases whose sizes vary from 147 to 268 rules. All of the results were obtained from running the Rule Base Clusterizer (RBC) tool that implements the algorithms explored in the previous sections of this paper.

4.2. 'BACTEREM' rule base

The BACTEREM rule base is from a KBS for medical diagnosis which was developed for use in hospitals in Israel (Ein-Dor, 1996). It has 268 rules and was built using the VP-Expert shell. Figure 3 shows the result of juxtaposing the BACTEREM rule base. The vertical scale is a dual one both for the Miller% (dashed line) and for the entropy (solid line). The horizontal scale is based on 'streamlining' from the sorted set of possible criterion values for the rule base being maintained. As pointed out in Section 2.4, the Hopfield net remains fixed between change points. Therefore computations are speeded up (i.e., 'streamlined') when entropy and Miller% are computed only at the change points.

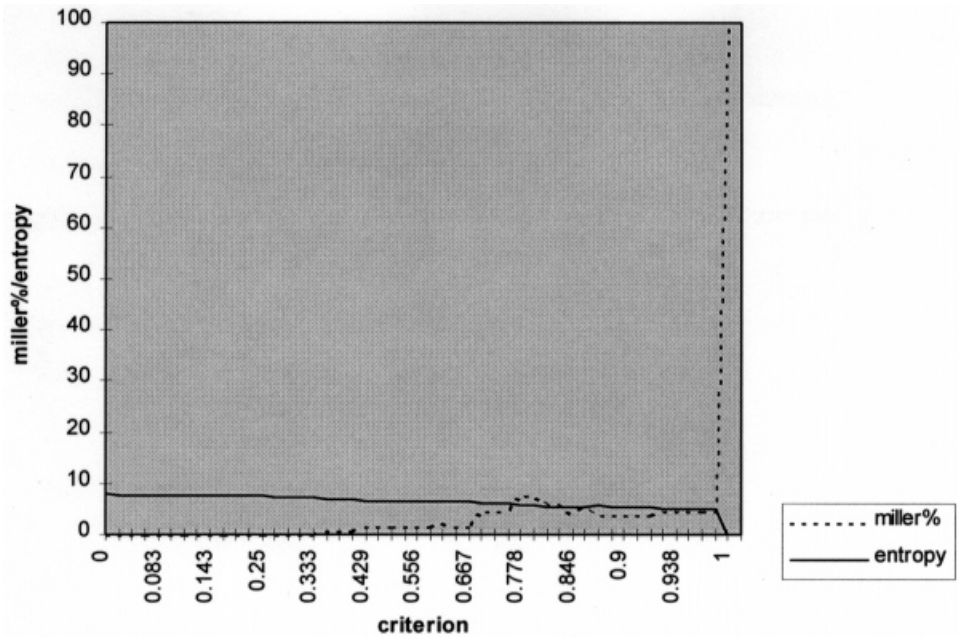

Figure 3. 'BACTEREM' rule base juxtapositioning

The BACTEREM rule base is not homogeneous. Rather it has a diversified rich vocabulary largely due to its medical diagnosis domain where words describing symptoms and diseases must be used. Streamlining was used to determine the change points for the criterion values. For a more focused look at the graph, Figure 4 shows a plot for criterion values only from 0 to 0·963 and entropy and Miller% values less than 10.

This rule base has three points of intersection at the criterion values of 0·75, 0·846 and 0·963. At these three values, the entropy values are, respectively, 6·15, 5·493, 4·858. We chose the intersection point with the criterion values of 0·75 because it has the highest entropy value at 6·15 even though the Miller% value is 4·55%. Notice that at the criterion value 0·75, the actual intersection does not occur; there is no actual criterion value corresponding to the intersection of the entropy and Miller% values even though, in the graph, it looks like the intersection exists; in other words, since the horizontal scale (change points of criterion values) is discrete, and the graph is, in fact, a discrete one with the points connected. And this is the reason that the values of entropy and Miller% are different. Thus we choose 0·75 since it is the best approximation according to the strategy for choosing the intersection point as described in Section 3.5.

4.3. 'CONTRACT' rule base

The CONTRACT rule base is from a KBS which is used for selecting contractors for construction work (Hicks, 1996). It has 147 rules and was built using the VP-Expert

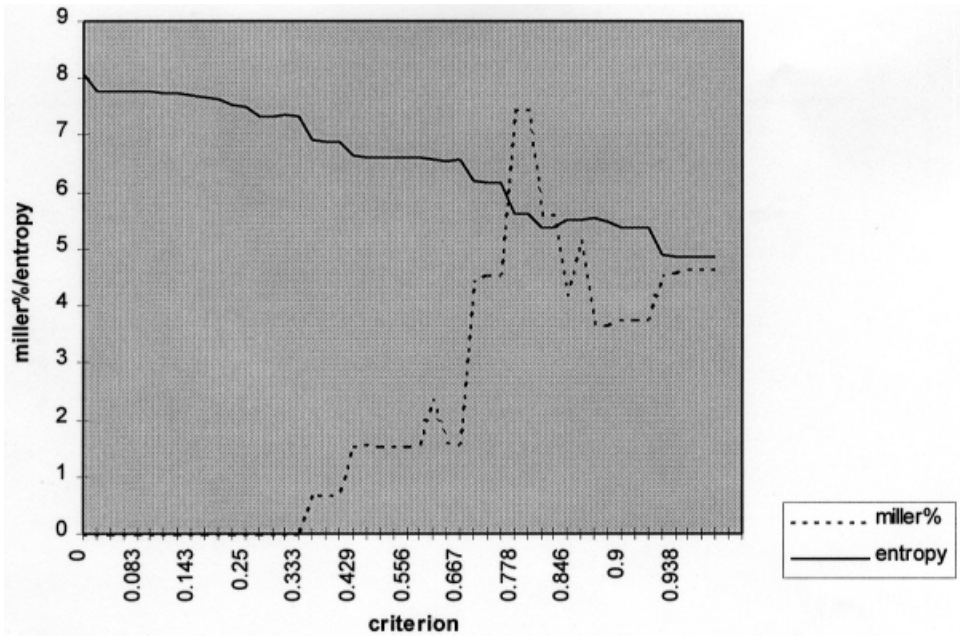

Figure 4. 'BACTEREM' rule base juxtapositioning (focused)

shell. Figure 5 shows the result of juxtaposing the CONTRACT rule base on the criterion values obtained by interpolating rule distances.

For a more focused look at the graph, we plot criterion values only from 0 to 0·875 and Miller% and entropy values of less than 30, as shown in Figure 6.

This rule base is quite homogeneous in that it uses the same terms over and over again in many different rules, resulting in a low vocabulary size. This result is largely due to the contracting domain where a few words describe a contractor's equipment status, skill level, reliability, safety, etc. The intersection point is 0·286 whose entropy value is 4·88 and Miller% value is 3·23%.

4.4. 'ADVICE' rule base

The ADVICE rule base is from a KBS for advising foreign students in selecting American graduate schools (Lin, 1996). It has 172 rules and was built using the VP-Expert shell. Figure 7 shows the result of juxtaposing the ADVICE rule base on the criterion values which are obtained by streamlining rule distances.

This rule base is extremely homogeneous. It uses the same terms repetitively in many different rules, resulting in a very small vocabulary size. The domain, advising foreign students in selecting US universities, requires few words to describe the student's preference such as weather, cost, and the student's score on the GRE and TOEFL examinations. The intersection point is 0·125 whose entropy value is 3·87 and Miller% value is 25·8%. Note that if a criterion value = 0·111 (entropy value is 7·36 and Miller% value is 0%)

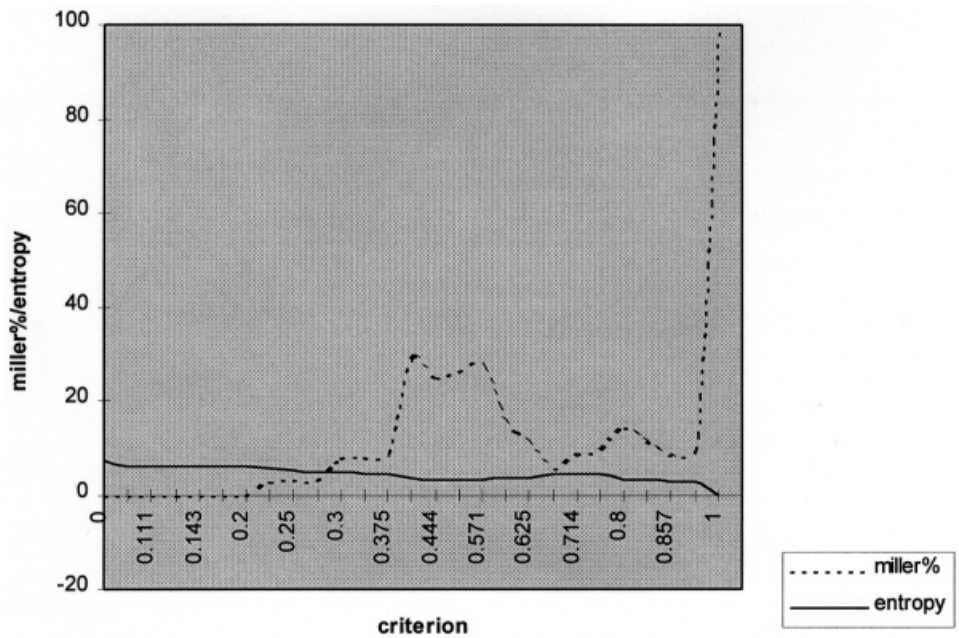

Figure 5. 'CONTRACT' rule base juxtapositioning

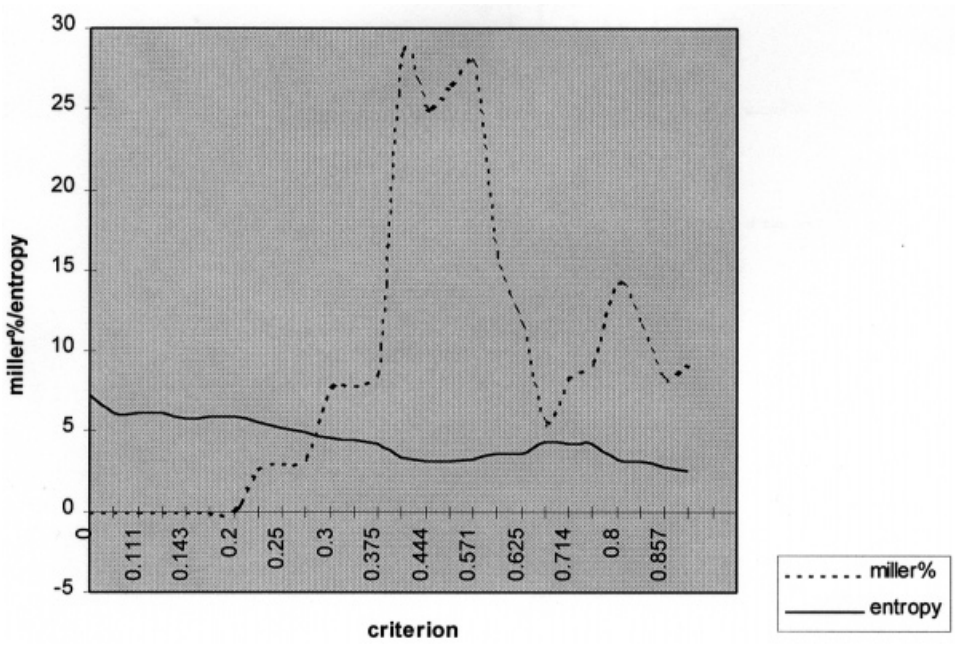

Figure 6. 'CONTRACT' rule base juxtapositioning (focused)

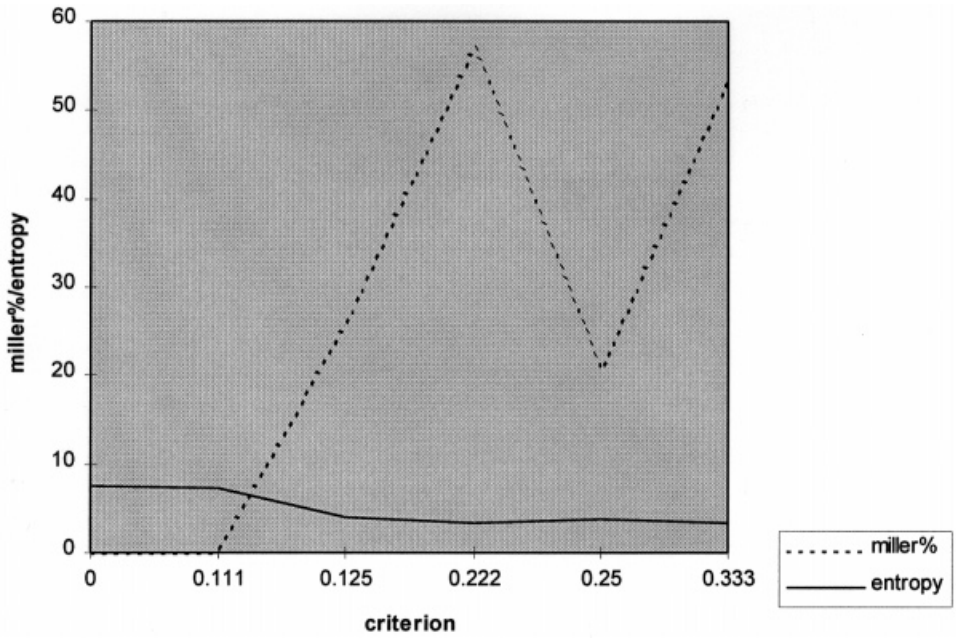

Figure 7. 'ADVICE' rule base juxtapositioning

was chosen, it would yield a higher entropy. However, because at 0·111 there is very little clustering done to the rule base, almost every rule is its own cluster. Thus we choose the change point at 0·125 as the criterion value.

5. CONCLUSIONS

5.1. Summary of findings

The maintainer of a knowledge-based system (KBS) faces a more difficult task than does the maintainer of a conventional system. KBS rule bases are usually unstructured. As a result, they are difficult to understand and hence difficult to maintain. To make the maintenance easier, a tool is needed to help the maintainer understand the KBS rule base. The tool should be based solely on the lexical information in the rule base in order for it to be applicable independent of the rule base domain. The concept of the Hopfield neural net was adapted to create clusters of rules. The clusters depend only on the choice of a parameter (criterion value) between 0 and 1 inclusive. The choice of criterion value involves a trade-off between the amount of information that clusters provide (measured in terms of entropy) and the limits of human cognition (measured in terms of Miller's magic number of 7 ± 2).

Using these concepts, we created a software tool to assist in the maintenance of KBS rule bases, the Rule Base Clusterizer. This tool aids in conducting maintenance actions by reducing time and effort in locating relevant rules. In other words, when rules are in the same cluster, they are likely to have many common terms and have logical connections. Consequently, a change on a certain rule in a cluster means additional changes on other rules in the rule base and especially on those rules in the same cluster. The maintainer can find these relevant rules in the cluster without having to search the entire rule base. We report in this paper on applying the Rule Base Clusterizer to three real-world KBS rule bases.

5.2. Limitations of the research

The four limitations on the research results described here are indicative of the range of limitations on the research done.

(1) The research reported here is concerned with maintaining knowledge-based systems which use rules as the method for knowledge representation. Creating clusters of rules is not of help in maintaining systems that use other forms of knowledge representation, such as frames.
(2) The testing of the Rule Base Clusterizer was done on rule bases implemented in VP-Expert. Even though rule bases implemented in other languages use the same IF . . .THEN. . . style, testing has not been done on rule bases implemented in other languages.
(3) The present computer implementation of the Rule Base Clusterizer is inefficient. In this paper, no effort was made to reduce the burden of the computing job

significantly by taking advantage of the structure of the distance and Hopfield matrices with their 0's on the diagonal and their symmetry between the upper and lower triangles. The magnitude of the computer limits can be seen from the following rough calculation. A Hopfield matrix consists of the values 0, 1 and −1. Thus, it takes a minimum of two binary bits to represent each Hopfield matrix element. For example, if a rule base has 1024 ($=2^{10}$) rules, the matrix has slightly over a million elements ($=2^{20}$). With two bits per element in one byte, approximately 2MB are required to store just the Hopfield matrix. Still more storage is needed for other matrices and for working space. Using brute force techniques lengthen the CPU time for executing the algorithms. Reductions could be obtained by using the symmetry between the upper and lower triangles of the matrix, for example.

(4) The reported results were obtained by computer analysis and test runs. Although it is reasonable, based on our analysis, to assume that the tool will help KBS rule base maintainers in their work, this assumption was not tested in the field or in the laboratory.

5.3. Directions for future work

Each of the limitations listed above implies the need for further work. The following specific studies are recommended.

(1) The underlying concept of this research is that clustering helps the maintainer. To validate this concept, empirical research should be conducted in which the Rule Base Clusterizer is used by KBS rule base maintainers. Such research could be done well in a controlled laboratory or field setting. The experiments could involve two sets of maintainers, one using the Rule Base Clusterizer and one without. The two groups should be matched for skill and experience level. Rule bases of increasing size and complexity would be given to members of each group. The hypotheses that people with the tool perform higher quality maintenance and require less time for their task would be tested.

(2) Assuming that the empirical results support the concepts of this research, the next step is to improve the sophistication of the computations in the Rule Base Clusterizer so that 'medium' and 'large' rule bases can be handled by the tool within a reasonable processing time on available computer hardware. The objective would be to find computational methods that overcome the size limits of the current work.

(3) Assuming that item 1 is successful, and in parallel with item 2, the Rule Base Clusterizer should be expanded so that it can handle frames and files from languages such as Lisp, Prolog and C++. This would extend the usefulness of the tool to a greater diversity of knowledge-based systems.

Acknowledgements

We thank Dr Trevor Bench-Capon of the University of Liverpool who gave insightful advice during the process of revising this paper, and Dr Phillip Ein-Dor of Tel-Aviv University and Claremont Graduate University for his support and encouragement during Ook Lee's doctoral dissertation work from which this paper was drawn.

References

Ahituv, N. and Neumann, S. (1990) *Principles of Information Systems for Management*, Third Edition, William C. Brown, Inc., Dubuque IA, 653 pp.

Chen, H., Hsu, P., Orwig, R., Hoopes, L. and Nunamaker, J.F. (1994) 'Automatic concept classification of text from electronic meetings', *Communications of the ACM*, **37**(10), 56–73.

Coenen, F. and Bench-Capon, T. (1993) *Maintenance of Knowledge-based Systems*, Academic Press, Inc., Troy MO, 322 pp.

Ein-Dor, P. (1996) *BATEREM*, Unpublished Computer Software, School of Management, Tel Aviv University, Tel Aviv, Israel.

Fausett, L. (Ed.) (1994) *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice-Hall, Inc., Englewood Cliffs NJ, 461 pp.

Hicks, R. (1996) *CONTRACT*, Unpublished Computer Software, Department of MIS, College of Business, University of Nevada at Las Vegas, Las Vegas NV.

Lin, M. C. (1996) *ADVICE*, Unpublished Computer Software, School of Education, University of Missouri at Columbia, Columbia MO.

Miller, G. (1956) 'The magical number seven, plus or minus two: some limits on our capacity for processing information', *The Psychological Review*, **63**(2), 81–97.

Pigford, D. V. and Bauer, G. (1995) *Expert Systems for Business: Concepts and Applications*, Boyd & Fraser Publishing Co., Boston MA, 447 pp.

Sharon, D. (1996) 'Meeting the challenge of software maintenance', *IEEE Software*, **13**(1), 122–125.

Soloway, E., Bachant, J. and Jensen, K. (1987) 'Assessing the maintainability of XCON in RIME: coping with the problems of a very large rule-base', in *Proceedings of the National Conference on Artificial Intelligence (AAAI 87)*, MIT Press, Cambridge MA, pp. 824–829.

von Mayrhauser, A. and Vans, A. M. (1995) 'Program comprehension during software maintenance and evolution', *IEEE Computer*, **28**(8), 44–55.

Authors' biographies:

Ook Lee is a Professor of Information Systems in the Department of Business Administration at Hansung University in Seoul, Korea. Previously, he worked as a Project Director at Information Resources, Inc., in Chicago Illinois, and as a Senior Information Research Scientist at Korea Research Information Center in Seoul, Korea. His main research interests include expert systems, neural networks, software engineering, digital libraries, electronic commerce and critical social theory. He holds a B.S. in Computer Science and Statistics from Seoul National University in Seoul, Korea, and an M.S. in Computer Science from Northwestern University in Evanston Illinois. He also earned an M.S. and Ph.D. in Management Information Systems from Claremont Graduate University in Claremont California. His email address is leeo@hansung.ac.kr

Paul Gray is a Professor in and the Founding Chair of the Department of Information Science at Claremont Graduate University in Claremont California. He specializes in decision support systems and data warehousing. In addition to 18 years in research and development organizations, including nine years at SRI International, he has served as a faculty member at Stanford University, Georgia Institute of Technology, University of Southern California and Southern Methodist University. In 1992–1993, he was the President of The Institute of Management Sciences. Paul's Ph.D. is from Stanford University. His email address is Paul.Gray@cgu.edu

